Lab 3 Report: Wall Follower

Team 1

Shrika Eddula
Sammy Krem
Pyae Sone Nyo Hmine
Sydney Matthews

6.4200

March 15, 2025

1 Introduction: Sammy Krem, Shrika Eddula

Wall following is a fundamental robotics problem where a robot navigates by maintaining a specific distance from a wall using sensors. This capability is important for autonomous navigation, enabling a robot to reach its goal while avoiding obstacles. A prior lab simulated wall following to visualize the problem, but real-life implementation provides tangible applications without idealized dynamics. Unlike simulation, real constraints such as sensor noise, wheel slippage, and non-instantaneous braking must be addressed, making hardware implementation a more complex challenge.

Lab 3: Wall Follower builds on robotics concepts from the prior simulation-based lab, such as control and sensing. It also integrates key robotics tools like Robot Operating System 2 (ROS2), a framework for robotics development, and NumPy, a library for mathematical operations. This lab bridges software and hardware by applying these concepts to a medium-sized remote controlled (RC) car equipped with an Nvidia computer and various sensors, allowing it to follow a wall efficiently without collisions or unnecessary detours.

To achieve the lab's objectives, our team designed a wall-following algorithm using Python and ROS. Our approach utilizes the car's LiDAR sensor and a proportional-derivative (PD) control algorithm to maintain a specified distance from the wall, ensuring safe, reliable, and robust navigation.
Experimental results show that the wall follower consistently follows the wall without collision, even during some sharp turns, while maintaining smooth navigation. Additionally, results prove that the wall follower is robust and reliable across a range of maneuvers, speeds, and environmental changes.

2 Technical Approach: Shrika Eddula, Sydney Matthews

Our robotic system composed of two critical functional components: the *Safety Controller* and the *Wall Follower*. Both components were designed and implemented as independent ROS2 nodes, each responsible for distinct aspects of autonomous navigation and collision avoidance. The primary objective in this division was to ensure safe and robust autonomous navigation in a constrained environment while maintaining efficiency and adaptability in real-time robotic perception and control.

The *Safety Controller* is responsible for monitoring sensor data and issuing emergency stop (e-stop) commands when necessary to prevent collisions. It achieves this by dynamically adjusting its stopping distance based on the current speed of the robot, thereby ensuring a balance between responsiveness and safety. On the other hand, the *Wall Follower* is designed to maintain a desired distance from a wall while navigating along its contour. This is achieved through real-time LiDAR-based perception, line-fitting techniques, and a PID control algorithm that determines appropriate steering adjustments.

2.1 Safety Controller

2.1.1 Overview

The Safety Controller is a fundamental component in ensuring the autonomous robot's safe navigation by preventing potential collisions. Its primary function is to analyze LiDAR data in real time and issue emergency stop commands if an obstacle is detected within a dynamically computed stopping distance. To achieve this, the Safety Controller monitors both the robot's velocity and its environment, computing a safety threshold that varies proportionally with speed. The system relies on a combination of numerical computations and threshold-based heuristics to achieve fast and reliable emergency response.

2.1.2 Technical Implementation

The Safety Controller is implemented as a ROS2 node using Python and the `AckermannDriveStamped` and `LaserScan` messages, and it follows a modular design with three core functionalities: LiDAR data processing, dynamic emergency stop calculation, and command execution.

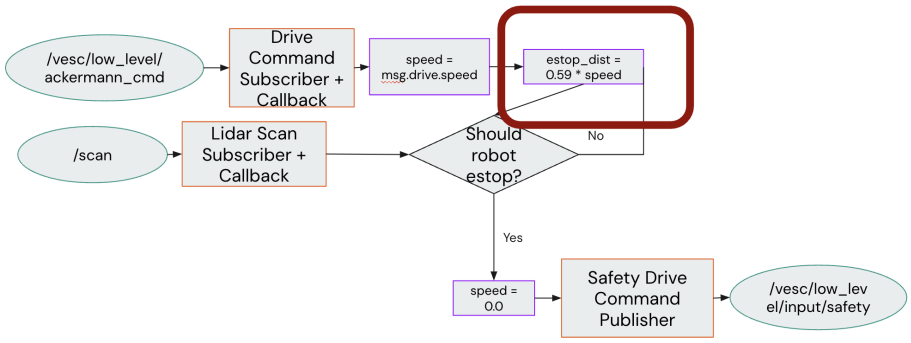


Figure 1: Safety Controller Flowchart: This diagram depicts the Safety Controller, showing the process of emergency stopping based on LiDAR input and velocity-dependent stopping distance calculations.

LiDAR Data Processing The LiDAR sensor provides a 240-degree scan of the surroundings, but for efficient processing and reducing unnecessary computations, the Safety Controller focuses only on a narrow 30-degree field of view directly in front of the robot. The system extracts LiDAR range values corresponding to this region:

$$\textbf{ranges} = \{r_i \mid i \in [-9^\circ, 9^\circ]\} \tag{1}$$

These range values are stored as a NumPy array for efficient vectorized operations. To determine whether an obstacle poses an immediate threat, the system checks if at least seven LiDAR points fall within the dynamically calculated emergency stop distance:

$$\text{ranges\_satisfied} = \sum (r_i < d_{\text{stop}}) \tag{2}$$

Emergency Stop Distance Calculation The stopping distance is a function of the robot's velocity. A linear scaling factor is applied to compute the required stopping distance:

$$d_{\text{stop}} = \alpha v \tag{3}$$

where v represents the current speed of the robot, and $\alpha = 0.59$ is an empirically determined coefficient that ensures safe braking.

Emergency Stop Execution If the system detects an obstacle within the emergency stopping range, it initiates a two-step braking sequence. First, it commands the robot to stop completely:

$$v_{\text{command}} = 0.0 \tag{4}$$

After a short delay, the robot remains stationary, ensuring that it has come to a complete stop.

2.1.3 Design Considerations

Several key design choices were made to optimize the Safety Controller:

- **Computational Efficiency:** Processing a limited LiDAR range reduces computational overhead.
- **Adaptability:** The stopping distance dynamically scales with velocity, making the system responsive to speed changes.
- **False Positive Mitigation:** The system requires multiple LIDAR points to confirm an obstacle, reducing false activations.

2.2 Wall Follower

2.2.1 Overview

The Wall Follower component is designed to maintain a fixed distance from a wall while allowing the robot to navigate along its contour. This is accomplished by continuously analyzing LiDAR data, fitting a linear model to detect wall orientation, and using a PD controller to correct deviations from the desired distance. It is implemented as a ROS2 node using Python and the `AckermannDriveStamped` and `LaserScan` messages

2.2.2 Technical Implementation

LiDAR Data Processing The system first filters LiDAR data to select points corresponding to the chosen wall-following mode. The angular ranges are defined as:

$$\text{Left Wall: } \theta \in [-25^\circ, 135^\circ] \tag{5}$$

$$\text{Right Wall: } \theta \in [-135^\circ, 25^\circ] \tag{6}$$

Wall Detection Using Line Fitting To determine the robot's distance from the wall, LiDAR points are transformed into Cartesian coordinates:

$$x = r\cos(\theta) \tag{7}$$
$$y = r\sin(\theta) \tag{8}$$

A NumPy-based linear regression model is applied to fit a straight line:

$$y = mx + b \tag{9}$$

The perpendicular distance from the robot to the wall is then computed as:

$$d = \frac{|b|}{\sqrt{m^2 + 1}} \tag{10}$$

PD Controller A Proportional-Derivative (PD) controller combines two control terms: proportional and derivative. The proportional term, $K_p e$, adjusts the system based on the current error $e = d - d_{\text{desired}}$, where d is the current distance and d_{desired} is the target distance. The derivative term, $K_d \frac{de}{dt}$, helps to anticipate future errors by considering how fast the error is changing, providing damping to reduce overshoot. The control law for the steering angle θ_{steer} is given by:

$$\theta_{\text{steer}} = K_p e + K_d \frac{de}{dt}$$

Together, these terms help to adjust the robot's behavior by ensuring both quick response and stability.

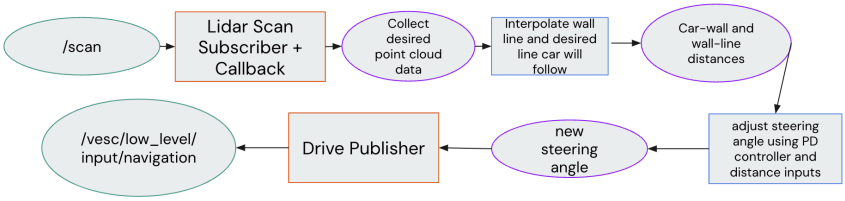


Figure 2: Wall Follower Flowchart: This diagram illustrates the Wall Follower logic, where the LiDAR scan data is used to interpolate wall distances and adjust steering using a PD controller.

A proportional-derivative (PD) controller then computes the steering angle:

$$\theta_{\text{steer}} = K_p e + K_d \frac{de}{dt} \tag{11}$$

where $K_p = 4$ and $K_d = 4$ are empirically tuned gains.

2.2.3 Design Considerations

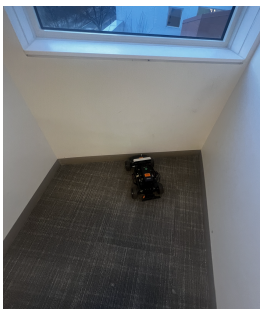
- **Robustness:** The linear regression model smooths noisy LiDAR data and interpolates a line that represents the wall.
- **Real-Time Adaptability:** ROS2 parameters allow for dynamic adjustments.
- **Efficiency:** NumPy is used for optimized numerical computations.

3 Experimental Evaluation: Sammy Krem, Pyae Sone Hmine

The wall-following race car was evaluated with a rigorous set of tests: different angled turns, speeds, and starting distances from the desired line. These wall-following experiments were successfully reproduced in various environments, including a spacious carpeted classroom with tables and chairs, a university basement with large intersecting hallways and pillars, and a dormitory with narrower hallways and lounge areas. The robot was also assessed in realistic scenarios, such as braking at an artificial dead end and evaluating the controller's response. This evaluation aimed to assess the robot's wall-following and safety controller performance across a variety of conditions.

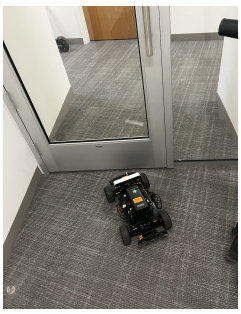
Safety Controller Braking Success at Different Speeds

The safety controller was tested at different speeds with a fixed desired distance of 0.6 meters. The robot drove into three narrow dead ends, where turning was not feasible given its proximity to the wall. Six trials were conducted at each speed, across three locations, and reproduced twice. A failure was deemed as hitting the wall, while a success was stopping before hitting it.



((a)) Race car, traveling at 1.2 m/s, hits Dead End 1's wall

((b)) Race car successfully braking at dead end 2



((c)) Race car successfully breaks in dead end 3

Figure 3: Safety controller is generally robust to different dead ends.

Table I: Success rate at different speeds.

Speed (m/s)	**Success Rate (6 Trials)**
1.2	66.66%
1.0	100%
0.8	100%
0.6	100%

As shown in Table I, the RC car successfully followed the wall at speeds of 1.0 m/s or lower 100 percent of the time, demonstrating the algorithm's reliability and safety at moderate speeds. However, at 1.2 m/s, failures or near-wall contact occurred due to an insufficient emergency stop distance constant. Despite this, the algorithm effectively prevents crashes, ensuring safe operation under most conditions.

Other failures of the safety controller occurred in environments with narrow hallway entrances and cluttered classrooms. The robot's 18-degree field of view limited its ability to detect obstacles, like chair legs, unless they were directly in front of it. In cluttered rooms, the robot failed to react to turns promptly, triggering the safety controller too late. However, in open environments with more predictable wall geometry, the robot remained a safe distance from the wall.

Figure 4: Failure modes in different environments.

Speed vs. Controller Response

During testing, the robot's PD and LiDAR-based controller demonstrated reliable and stable performance, correcting deviations smoothly across different speeds over multiple trials. The robot consistently maintained an optimal path along the wall, with an initial parallel distance of 0.304 meters to the right wall and a target following distance of 0.75 meters. A performance trial was evaluated over a six-meter distance, and similar results were reproduced in other trials.

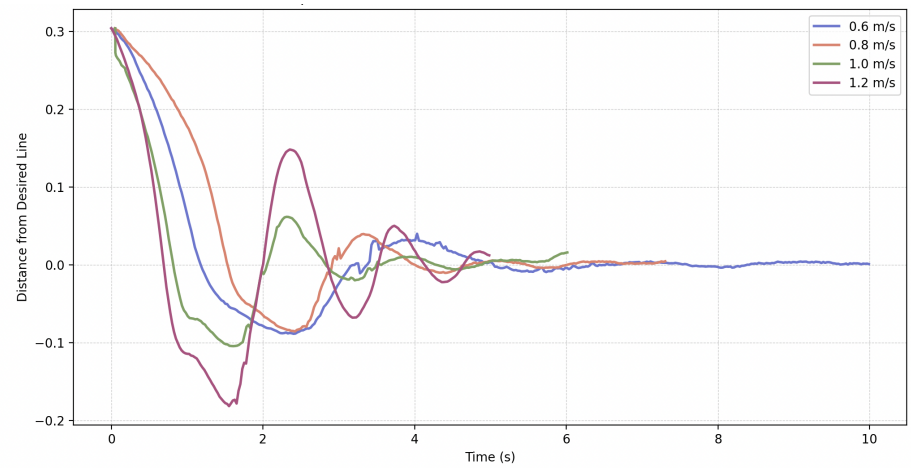


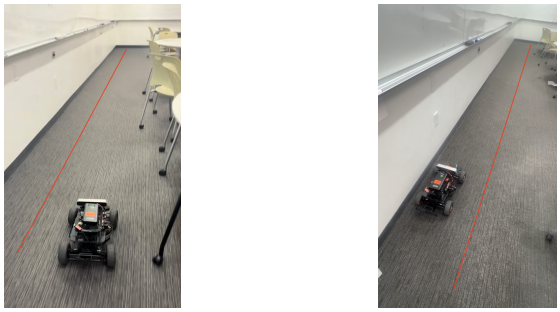
Figure 5: The controller demonstrated quick convergence, consistently achieving steady-state within 0.02 meters (2.6% tolerance) in 4.6 seconds. However, as speed increases, oscillations intensify due to the stronger response of the derivative term in the PD controller. While the system still converges rapidly, the increased oscillation reduces stability, extends the settling time, and makes the overall response less desirable at higher velocities.

Success of Turns and Maneuvers

The robot was evaluated for its ability to maintain a smooth trajectory while navigating different turn angles and starting distances. Successful maneuvers were characterized by minimal oscillations, maintaining a trajectory within 0.1 meters of the desired line within 5 seconds of the maneuver, and completing turns as expected. Turns and maneuvers are classified as follows:

- **Perpendicular Turn** ($90°$): A sharp right-angle turn.
- **Acute Angle Turn** ($< 90°$): A tight turn requiring quick adjustments.

- **Obtuse Angle Turn** ($> 90°$): A gradual turn, easier for the robot to handle.



((a)) Positive Starting Distance from Line

((b)) Negative Starting Distance from Line

Figure 6: The RC car rapidly adapts to maintaining the desired distance from the left wall.

Table II: Success rate for different maneuver types.

Maneuver Type	**Success Rate (10 Trials)**
Obtuse Angle Turn	100%
Perpendicular Turn	100%
Acute Angle Turn	30%
Negative Starting Distance from Desired Line	100%
Positive Starting Distance from Desired Line	100%

As seen in Table II, failures were most prominent with acute angles, where the robot's turning radius exceeded the available space (requiring a large change in steering angle in little time which is difficult to do physically), triggering the safety controller to stop the robot. Despite this, the robot maintained a 100 percent success rate and was robust for positional deviations from the desired line and other angle classifications, indicating that minor to moderate adjustments did not impact performance.

Figure 7: The wall follower detects the flower plot ahead of time so it can drive around it.

Other miscellaneous successes of the wall follower include avoiding larger obstacles, such as a flower pot, collision free. This miscellaneous example that the wall follower is robust to every day scenarios.
Below are links for videos of the robot driving: https://drive.google.com/file/d/1rUn2YMGfuNLyRf6xvyCTDCEWWP0I
https://drive.google.com/file/d/1pR$_X$3$XD7mg6riTiA5wCi0yqjFRugOWHV/view?usp = sharing$
https://drive.google.com/file/d/1br1iWGXuCgypyNNM0LsB8RI6pfkJuCij/view?usp=sharing

4 Conclusion: Pyae Sone Hmine, Sydney Matthews

The Wall Follower lab successfully demonstrated the application of fundamental robotics concepts in real-world scenarios, bridging the gap between simulation and hardware implementation. By integrating ROS2, LiDAR-based perception, and a PD control system, our team was able to design a robust wall-following algorithm that maintained a desired distance from the wall while adapting to various environmental conditions. The addition of a Safety Controller ensured collision prevention, enhancing the overall reliability of the system.
Through extensive testing, we observed that the system performed well across different turn types, with the highest success rates recorded for obtuse and perpendicular turns. However, acute turns presented a greater challenge, highlighting the need for further tuning of the control algorithm to improve stability and reduce oscillatory behavior at higher speeds. Additionally, instances where the robot was unable to execute sharp turns due to physical constraints underscored the importance of path-planning improvements.
Future work could focus on optimizing the PID tuning for better responsiveness, incorporating more advanced predictive control techniques, and improving maneuverability in tight spaces. Overall, this lab provided valuable hands-on experience with autonomous navigation, reinforcing key concepts in perception, control, and real-time decision-making for mobile robots.

5 Lessons Learned

Shrika: During this lab, I learned the importance of efficiently processing LiDAR data in real-time, particularly how narrowing the field of view and leveraging NumPy for vectorized operations

significantly reduces computational overhead. Implementing the Safety Controller taught me how to dynamically adjust stopping distances based on velocity, and I also realized the trade-offs between empirical tuning (using a coefficient like 0.59) versus a physics-based approach ($\frac{v^2}{9.81}$) for emergency stops. Additionally, I learned that clear communication in team-based development is crucial—documenting ROS topics, defining well-structured callback functions, and ensuring consistent parameter naming helped streamline debugging and integration between our safety and navigation modules.

Pyae Sone: This lab exposed me to the fundamental skillsets in implementing theoretical concepts learned in class. One of the most important skills we learned was integration, logging our progress, and figuring out simple and effective approaches to implementation. In technical details, I learned much more deeply how ROS works with the different components of the racecar and how they communicate with each other through nodes and packages. I also understand the dynamics of the system in a more formal approach such as Ackerman drive and Bicycle dynamics. I also learned various skills in working with the team, such as breaking down the problem into simpler chunks and communicating the idea well to my teammates. This allowed us to have great success in our teamwork.

Sydney: Throughout this lab, I learned about the process required to implement a program on a physical robotic system. Taking our simulation from last lab and running it on our robot involved many more systems than I was expecting, and we had to account for many factors that were not replicated in the simulation: friction, hardware performance issues, etc. As for non-technical lessons, I learned the importance of effective communication among team members and how to split up tasks to accomplish our goal altogether. Additionally, I now understand the importance of debugging and testing code throughout the process of writing a program.

Sammy: The most important technical concept I learned during this lab was how to debug code effectively. I gained more exposure to debugging ROS package management, specifically learning how to properly create ROS packages and knowing what to fix when something like a build terminal command doesn't work. I also learned how to debug issues where errors aren't thrown, such as when our robot wasn't moving but we thought the wall follower algorithm was setting the velocity. I learned that logging statements, and echoing ros topics, and drawing out flow charts for our process helps a lot with debugging. Two important communication skills I improved upon was learning when to ask for help and learning how to be more patient in team meeting when hardware issues and software bugs arise.